Bounding the Resource Availability of Partially Ordered Events with Constant Resource Impact

Jeremy Frank

Computational Sciences Division
NASA Ames Research Center, MS 269-3
frank@email.arc.nasa.gov
Moffett Field, CA 94035

Abstract. We describe a resource bounding technique called the *Flow Balance Constraint* (FBC) to tightly bound the amount of available resource for a set of partially ordered events with piecewise constant resource impact. We provide an efficient algorithm for calculating FBC and bound its complexity. We compare this technique with two existing resource bounding techniques, the Balance Constraint (BC) due to Laborie and the Resource Envelope (E_t) due to Muscettola. We prove that using FBC to halt search under chronological search with a static variable and value order generates smaller search trees than either BC or E_t. We also show that E_t and BC are not strictly comparable in terms of the size of the search trees generated under chronological search with a static variable and value order. We then show how to generalize FBC to construct tighter resource bounds but at increased computational cost.

1 Introduction

Scheduling requires ordering tasks while simultaneously satisfying temporal and resource constraints. Finding and maintaining *temporally flexible* schedules has numerous advantages over finding a fixed-time schedule. Usually, scheduling is performed assuming that the problem's characteristics are known in advance, do not change, and that the execution of the schedule is deterministic. However, these assumptions are often violated in practice. For example, if events do not take place exactly when they are scheduled, it may be costly to find a new schedule consistent with the actual event execution times [4]. Techniques such as that described in [5] make it possible to efficiently update the flexible schedule once the precise timing of events are known. A second advantage is that it can be less expensive to find flexible schedules because fewer decisions need to be made; thus, less search is necessary. This is true for simple temporal constraints, but the presence of resource constraints requires *efficient tight resource bounding techniques* to determine that the resource constraint is always or never satisfied.

It is straightforward to calculate the resource bounds of events whose execution time are fixed. The task becomes more difficult when activities or events are not fixed in time or are unordered, and more difficult still when events can have arbitrary impact on resources. In the context of constructive search, early detection of success or failure is often important to achieving good search performance. In this paper we focus on

the ability of resource bounding techniques to reduce the cost of constructive search algorithms by means of detecting success or failure.

Following Muscettola [1], Laborie [2] has provided a simple but expressive formalism for scheduling problems called *Resource Temporal Networks* (RTNs). In this paper we study RTNs consisting of a *Simple Temporal Network* (STN) as described in [3], constant resource impacts (either production or consumption) for events, and piecewise constant resource bounds; Laborie [2] refers to this subclass as $\langle \mathcal{R}, \mathcal{L}_{\mathcal{Q}}, STN^{\neq} \rangle$. There are two existing techniques for bounding resource availability in such RTNs; the Balance Constraint (BC) [6] due to Laborie, and the Resource Envelope (E_t) [1] due to Muscettola. These techniques are described in more detail in Section 2. BC features an efficient but loosely bounding approximation, while E_t is more costly but provides a tight bound (in all cases a schedule justifying the bound is proved to exist). Somewhat surprisingly, these techniques are not strictly comparable in terms of the size of the search trees generated under chronological search. We provide examples demonstrating this in Section 3. In Section 4 we describe the *Flow Balance Constraint* (FBC), a novel synthesis of these two approaches. In Section 5 we prove that FBC generates smaller proof trees than either BC or E_t under chronological search. In Section 6 we describe the complexity of a naive algorithm to calculate FBC. We then show how to calculate FBC incrementally, thereby reducing its computational cost. In Section 7 we then generalize FBC in order to construct even tighter resource bounds but at increased computational cost. Finally, in Section 8 we conclude and describe future work.

2 Previous Work

In this paper we will assume that time is represented using integers[1]. We will also assume that the resource impact of each event is known when solving begins. We will assume each RTN has only one resource, that there is one constraint $\leq r$, and one constraint ≥ 0 on the resource. We also assume that the lower bound of the scheduling horizon is 0, and that r units of resource are available at time 0.

The techniques we study are aimed at establishing *Halting Criteria* (HC) for chronological backtracking search algorithms schedulers that maintain temporal flexibility. By "halt" we mean identifying a leaf node in the chronological search tree. A scheduler can halt if the *Necessary Truth Criterion* [2, 8] (NTC), is satisfied, namely, if *all* feasible solutions of the STN also satisfy the resource constraints. The NTC is satisfied if the upper bound on the available resource is always below the maximum available resource r, and the lower bound on the available resource is always above 0. Schedulers can also halt if *no* feasible solution of the STN can satisfy the resource constraint [2]. This is true if the upper bound on the available resource is ever below 0, or the lower bound on the available resource is ever above r. Otherwise, the scheduler must continue search, i.e. the current state is an interior node of the search tree.

We will use the following notation: Let $\mathcal{N}$ be the set of all events of an RTN and $n = |\mathcal{N}|$. Let $X \in \mathcal{N}$; $c(X)$ denotes the resource impact of X. If $c(X) < 0$ X is said to be a *consumer* ; if $c(X) > 0$ then X is said to be a *producer*. As defined in [1], X *anti-precedes* Y if X must occurs at or after Y (i.e. $Y \leq X$). A *predecessor set* of a set of events $\mathcal{S}$ has the property that if $X \in \mathcal{S}$ then every event Y such that $Y \leq X$ is also in $\mathcal{S}$. A *successor set* of a set $\mathcal{T}$ has the property that if $X \in \mathcal{T}$ then every event Y such that $Y \geq X$ is also in $\mathcal{T}$. Let R be an RTN and let $A(R)$ be any procedure for evaluating the HC. If $A(R) = T$ then the HC is true and R has a solution. If $A(R) = F$, the HC is true and R has no solution. If $A(R) =?$ the HC is false and it is unknown whether R has a solution.

[1] This assumption can be relaxed, but leads to resource bounds holding over half-open intervals of time.

[2] One could refer to this as the Necessary Falsity Criteria (NFC).

2.1 The Balance Constraint

Laborie's *Balance Constraint* (BC) [6] calculates bounds on the maximum and minimum amount of a resource immediately prior to and immediately following the execution of an event X. The STN is partitioned relative to X into the following sets: Before $B(X)$, Before or Equal $BS(X)$, Equal $S(X)$, Equal or After $AS(X)$, After $A(X)$, and Unordered $U(X)$. Note that the predecessor set of X is $\{B(X) \cup BS(X) \cup S(X)\}$ and the successor set of X is $\{A(X) \cup AS(X) \cup S(X)\}$ These sets are then used to calculate the following quantities: $L^{<}_{min}(X)$ for the minimum available resource before X occurs, $L^{<}_{max}(X)$ for the maximum available resource before X occurs, $L^{>}_{min}(X)$, for the minimum available resource after X has occurred, and $L^{>}_{max}(X)$ for the maximum available resource after X has occurred. A sample calculation: let $P^{>}_{min}(X) = \{B(X) \cup BS(X) \cup \{V \in AS(X) \cup U(X) | c(V) < 0\}\}$ Then $\sum_{Z \in P^{>}_{min}(X)} c(Z)$. is a lower bound on $L^{>}_{min}(X)$. The other bounds can be constructed in a similar manner. There may be no schedule consistent with the temporal constraints resulting in these calculated resource availabilities. The computational complexity of BC is dominated by the maintenance of arc-consistency of the STN.

2.2 The Resource Envelope

The *Envelope Algorithm* E_t [1] finds the *envelope* of an RTN, that is, the functions from time to the maximum $L_{max}(t)$ and minimum $L_{min}(t)$ available resource. Events are partitioned into $C(t)$, those that must have occurred by t; $P(t)$, those that are permitted to occur at t, and $O(t)$, those that can't occur until after t. The maximum available resource at a time t, $L_{max}(t)$, corresponds to a schedule in which the events in $P_{max}(t) \subset \{P(t) \cup C(t)\}$ all occur before t. To find the set $\Delta P_{max}(t) \subset P(t)$ that contributes to $P_{max}(t)$, a maximum flow problem is constructed using all anti-precedence links derived from the arc-consistent STN. The rules for building the flow problem to find $L_{max}(t)$ are as follows: all events in $P(t)$ are represented by nodes of the flow problem. If $X \geq Y$ then the flow problem contains an arc $X \rightarrow Y$ with infinite capacity. If $c(X) > 0$ then the problem contains an arc $\sigma \rightarrow X$ with capacity $c(X)$. If $c(X) < 0$ then the problem contains an arc $X \rightarrow \tau$ with capacity $|c(X)|$. To construct the flow problem to find $L_{min}(t)$, if $c(X) > 0$ then the problem contains an arc $X \rightarrow \tau$ with capacity $c(X)$, and if $c(X) < 0$ then the problem contains an arc $\sigma \rightarrow X$ with capacity $|c(X)|$. Examples of the flow problem construction are shown in Figure 1. The maximum flow of this flow network matches all possible production with all possible consumption in a manner consistent with the precedence constraints. The predecessor set of those events reachable in the residual flow is $\Delta P_{max}(t)$, and $\Delta P^{C}_{max}(t) = \{P(t) - \Delta P_{max}(t)\}$. The tightness of the bound is guaranteed by proving that adding the constraints $\{X_{ub} \leq t\} \forall X \in \Delta P_{max}(t)$ and $\{Y_{lb} > t\} \forall Y \in \Delta P^{C}_{max}(t)$ is consistent with the original STN.

It turns out that $\Delta P_{max}(t)$ need only be computed at $\leq 2n$ distinct times. The complexity of the algorithm described in [1] is $O(n * MaxFlow(n, m) + n^2)$, where $n = |\mathcal{N}|$ and m is the number of anti-precedence relationships in the arc-consistent STN. In [7] this is reduced to $O(MaxFlow(n, m) + n^2)$ by taking advantage of the order in which edges are added to and removed from a single maximum flow problem. The crucial observation is that when computing E_t an event always move from open to pending to closed and stays closed. As shown in Figure 1 (b) and (c), this guarantees that events topologically "late" in the flow problem are removed, while topologically "early" events are added to the flow problem. The insight is that the flow problem need not be solved anew; the previous flow calculation can be reused, thereby saving a factor of n. As an aside, a more general incremental maximum flow algorithm is given in [9], but it doesn't assume any particular order of flow arc additions and deletions.

3 Examples of Non-Domination

Muscettola previously demonstrated that E_t can prove the HC is satisfied in cases where BC cannot. In this section we provide an example where E_t fails to prove the HC is satisfied in cases where BC can. The consequence of this is that neither algorithm "dominates" the other when used solely to halt search. Since E_t provides the tightest possible mapping from time to minimum and maximum resource availability, it is somewhat surprising that E_t fails to dominate BC.

3.1 BC Doesn't Dominate E_t

Muscettola [1] describes an RTN for which BC cannot not prove the HC is satisfied but E_t can. This example is modified and reproduced in Figure 1. Initially the resource has 2 units available. BC will calculate $L^{>}_{min}(A)$ as follows: $U(A) = \{W, X, Y, Z\}$ and $A(A) = \{B, C, D\}$. It assumes W and Y occur before A, and that X and Z occur after A. Since $L^{>}_{min}(A) = -1$, BC concludes that more decisions are needed. Clearly, this schedule is not consistent with the original STN, so the bound calculated by BC is "loose". The E_t algorithm is able to prove that the HC is satisfied in this case by determining $L_{min}(t) \geq 0$ and $L_{max}(t) \leq 2$ over the scheduling horizon. As a case in point, consider $L_{min}(10)$. The pending set $P(10) = \{B, C, D, X, Y, Z\}$ and the resulting flow problem is shown in Figure 1 (c). The maximum flow for this network saturates the arcs to C and Y. No events are reachable in the residual graph, so $P^{<}_{max}(10) = \emptyset$. We see that $L_{min}(10) = 0$ since A and W are closed at 10.

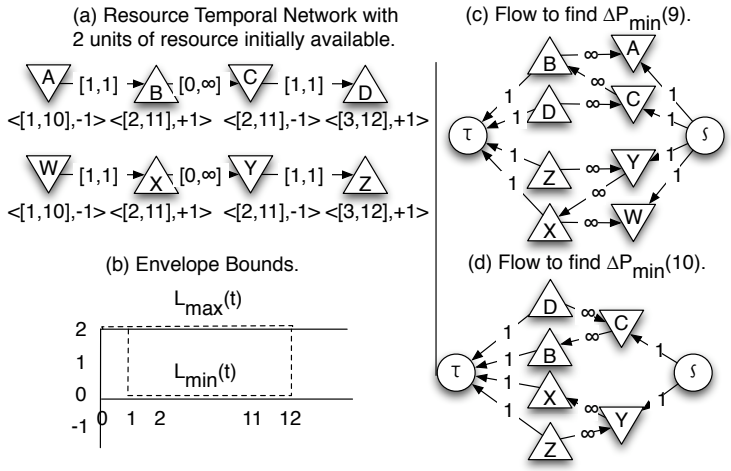

Fig. 1. An RTN for which BC fails to detect that the HC is true because a solution exists.

3.2 E_t Doesn't Dominate BC

Figure 2 describes an RTN for which BC can show that the HC is satisfied. Again, initially the resource has 2 units available. In this case, since $B(D) = \{A, B, C\}$ and $\{BS(D) \cup U(D)\} = \emptyset$, BC will find $L^{<}_{max}(D) = -5$. This proves that, no matter what additional constraints are imposed, the resource bound will be violated. By contrast, E_t cannot conclude that the HC is satisfied. We see that B and C can be postponed until time 10, and A can be scheduled as early as 1, leading to $L_{max}(1) = 3$. At time 2, D could occur, but maximizing resource availability would then require scheduling

everything before 2 with a resource availability of 0 being the result. At time 2 we can still find a schedule in which A occurs before all other events, with resource availability 3; so $L_{max}(2) = 3$. At time 10, however, both consumption events B and C must have taken place; in order to maximize the available resource, D must occur, leading to $L_{max}(10) = 0$. Furthermore, $L_{min}(t)$ provides no assistance in proving the HC is true. Not only does E_t fail to show that the HC is true, there are non-trivial ordering decisions that can be made; in the worst case, schedulers could spend a considerable amount of time fruitlessly continuing the search from states like the one shown in Figure 2.

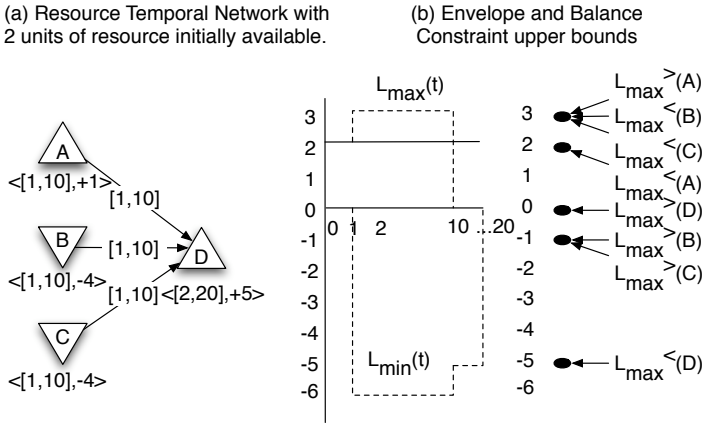

Fig. 2. An RTN for which E_t fails to detect that the HC is true because no solution exists.

4 A Better Check for the Halting Criteria

In order to halt search more effectively than either E_t or BC, we adopt a synthesis of both strategies. We use the same partition of events used by Laborie [6]. Like Laborie, we then find bounds on $L^{<}_{max}(X), L^{<}_{min}(X), L^{>}_{max}(X)$, and $L^{>}_{min}(X)$. Like Muscettola, we build a maximum flow problem whose residual graph is used to construct the supporting sets $P^{<}_{max}(X)$, $P^{<}_{min}(X)$, $P^{>}_{max}(X)$, and $P^{>}_{min}(X)$. The rules for constructing the flow problem are identical to those for calculating E_t; only the set of events defining the flow problems are different. To find $\Delta P^{<}_{max}(X)$ and $\Delta P^{<}_{min}(X)$, we solve a flow problem over the set of events $BS(X) \cup U(X)$. In these cases, $P^{<}_{max}(X) = \{\Delta P^{<}_{max}(X) \cup B(X)\}$ and $P^{<}_{min}(X) = \{\Delta P^{<}_{min}(X) \cup B(X)\}$. We define $\Delta P^{<C}_{max}(X) = \{\{BS(X) \cup U(X)\} - \Delta P^{<}_{max}(X)\}$ and $\Delta P^{<C}_{min}(X) = \{\{BS(X) \cup U(X)\} - \Delta P^{<}_{min}(X)\}$ To find $\Delta P^{>}_{max}(X)$ and $\Delta P^{>}_{min}(X)$, we solve a flow problem over the set of events $AS(X) \cup U(X)$. In these cases, $P^{>}_{max}(X) = \{\Delta P^{>}_{max}(X) \cup S(X) \cup B(X) \cup BS(X)\}$ and $P^{>}_{min}(X) = \{\Delta P^{>}_{min}(X) \cup S(X) \cup B(X) \cup BS(X)\}$. We define $\Delta P^{>C}_{max}(X) = \{\{AS(X) \cup U(X)\} - \Delta P^{>}_{max}(X)\}$ and $\Delta P^{>C}_{min}(X) = \{\{AS(X) \cup U(X)\} - \Delta P^{>}_{min}(X)\}$. These supporting sets define schedules that prove the bounds are tight. For example, the resulting set $P^{<}_{max}(X)$ defines an STN constructed by adding the constraints $\forall V \in P^{<}_{max}(X)\{V < X\}$ and $\forall V \in P^{<C}_{max}(X)\{X \leq V\}$. We call the new procedure the *Flow Balance Constraint* (FBC), since it combines the features of the Balance Constraint with the flow-based approach. For each event X we must calculate 4 bounds; however, the actual number of bounds calculations is $\leq 4n$, since we observe that the same bounds apply $\forall V \in S(X)$. The algorithm for FBC is described in Figure 3. As

is done in [6], we maintain the partition of the STN relative to each event X during the arc-consistency enforcement phase.

FBC($\mathcal{R}$)
Enforce Arc Consistency on $\mathcal{R}$
Maintain event-centric partitions
for each event $X \in \mathcal{N}$
Set up flow problems
Bound(X)
end for
end

Bound(X)
Find $\Delta P^{<}_{max}(X) \subset BS(X) \cup U(X)$
$L^{<}_{max}(X) = \sum_{V \in \{\Delta P^{<}_{max}(X) \cup B(X)\}} c(V)$
Find $\Delta P^{<}_{min}(X) \subset \{BS(X) \cup U(X)\}$
$L^{<}_{min}(X) = \sum_{V \in \{\Delta P^{<}_{min}(X) \cup B(X)\}} c(V)$
Find $\Delta P^{>}_{max}(X) \subset \{AS(X) \cup U(X)\}$
$L^{>}_{max}(X) = \sum_{V \in \{\Delta P^{>}_{max}(X) \cup B(X) \cup BS(X) \cup S(X)\}} c(V)$
Find $\Delta P^{>}_{min}(X) \subset \{AS(X) \cup U(X)\}$
$L^{>}_{min}(X) = \sum_{V \in \{\Delta P^{>}_{min}(X) \cup B(X) \cup BS(X) \cup S(X)\}} c(V)$
end

Fig. 3. A sketch of the Flow Balance Constraint.

We now prove that the resulting bounds on quantities like $L^{<}_{max}(X)$ are tight. To do so, we first show that there is at least one schedule justifying the bound, and then show that there is no better bound than that found using the flow problem.

Theorem 1. *Let R be an RTN and X be an event in R. Suppose $\Delta P^{<}_{max}(X) \neq \emptyset$. Let R' be the STN formed by adding the following constraints to R: $\{V < X\}$ for all $V \in \Delta P^{<}_{max}(X)$ and $\{X \leq W\}$ for all $W \in \Delta P^{<C}_{max}(X)$. Then R' has at least one temporally consistent solution.*

Proof. Since $V \in \{BS(X) \cup U(X)\}$ the imposition of a single constraint alone doesn't make the STN inconsistent. Imposing a constraint $V < X$ can only decrease V_{ub} or increase X_{lb}. Since $\Delta P^{<}_{max}(X)$ is a predecessor set, all $\{V < X\}$ can be imposed simultaneously without impacting consistency. Imposing a constraint $X \leq W$ can only decrease X_{ub} or increase V_{lb}. Since $\Delta P^{<C}_{max}(X)$ is a successor set, all $\{X \leq W\}$ can also be imposed simultaneously without impacting consistency. Finally, X is the only event whose bounds are acted on by both classes of constraint. Consider two events A,B. Since $A \in \Delta P^{<}_{max}(X)$ and $B \in \Delta P^{<C}_{max}(X)$ we know it can't be the case that $B < A$. But then either $A \leq B$ or A and B can be ordered in any way, and we already know X can be ordered any way with respect to A or B. Thus, $A < X \leq B$ is possible and no such ordering prevents other linearizations with respect to X. □

Theorem 2. *Let R be an RTN and X be an event in R. Suppose $\Delta P^{<}_{max}(X) \neq \emptyset$. Then $\sum_{V \in \Delta P^{<}_{max}(X) \cup B(X)} c(V)$ is the maximum possible value of $L^{<}_{max}(X)$.*

Proof. Since we construct the flow problems in exactly the same way as [1], we state this as a corollary of Theorem 1 of [1]. □

The proofs for the tightness of the bounds on $L^{>}_{max}(X), L^{<}_{min}(X)$ and $L^{>}_{min}(X)$, are similar.

4.1 Discovering Constraints With FBC

The calculation of FBC allows the inference of new bounds on temporal constraints and new precedence constraints on events in a manner similar to BC [6]. We know that

$L^{<}_{max}(X) = \sum_{V \in \{\Delta P^{<}_{max}(X) \cup B(X)\}} c(V)$. Suppose $L^{<}_{max}(X) > 0, r + \sum_{V \in B(X)} c(V) = d < 0$ and that there is a predecessor set $P \subset \Delta P^{<}_{max}(X)$ such that $\sum_{V \in P} c(V) < |d|$. If X occurs before or at the same time P ends, then the resource constraint will be violated. Let t be the earliest end time of all such sets. Then $t + 1$ is a valid new lower bound for X, i.e. $X_{lb} > t$. If $|\Delta P^{<}_{max}(X)| = k$, calculating $\sum_{V \in P} c(V)$ and the earliest end time t for every predecessor set of every event requires $O(k^2)$ since we already store the precedences to create the flow graphs. We can also tighten the bound on X_{ub}. Suppose there is a predecessor set P such that $\Delta P^{<}_{max}(X) \subset P \subset \{BS(X) \cup U(X)\}$ and $\sum_{V \in P} c(V) < |d|$. If X occurs at the same time or after P ends then the resource constraint will be violated. Let t be the latest ending time of all such sets. Then $t - 1$ is a valid upper bound on X, i.e. $X_{ub} < t$. If $|\Delta P^{<C}_{max}(X)| = k$, calculating $\sum_{V \in P} c(V)$ and the earliest end time t for every predecessor set of every event requires $O(k^2)$. Similar arguments allow bounding X using $\Delta P^{<}_{max}(X)$ when $\sum_{V \in B(X)} c(V) > r$ as well as $\Delta P^{>}_{min}(X)$ and $\Delta P^{>}_{max}(X)$.

Inferring precedence constraints is a little trickier. Again, suppose $r + \sum_{V \in B(X)} c(V) = d < 0$. Imposing the constraint $X < Y$ doesn't impact $B(X)$ but may lead to a new set defining the maximum available resource, denoted $\Delta P'^{<}_{max}(X)$. Suppose that $\exists Y \in BS(X) \cup U(X)$ such that $X < Y \Rightarrow r + \sum_{V \in \{\Delta P'^{<}_{max}(X) \cup B(X)\}} c(V) < 0$. Then we can safely impose the precedence constraint $X \geq Y$. It is sufficient to find $Y \in \Delta P^{<}_{max}(X)$ such that its successor set $S \subset \Delta P^{<}_{max}(X)$ has the property that $\sum_{V \in \{\Delta P^{<}_{max}(X) - S\}} c(V) < |d|$. Under these circumstances, if X occurs before Y, then the remaining events in $\Delta P^{<}_{max}(X)$ cannot offset the events in $B(X)$.

Imposing the constraint $Y < X$ may lead to a new $B'(X)$. Suppose that $\exists Y \in \{BS(X) \cup U(X) - \Delta P^{<}_{max}(X)\}$ such that $Y < X \Rightarrow r + \sum_{V \in \{\Delta P'^{<}_{max}(X) \cup B'(X)\}} c(V) < 0$. Then we can safely impose the constraint $Y \geq X$. It is sufficient to find Y such that its predecessor set $S \subset \{BS(X) \cup U(X)\}$ has the property that $r + \sum_{V \in \{B(X) \cup S\}} c(V) = e < 0$ and $\sum_{V \in \{\Delta P^{<}_{max}(X) \cap S\}} c(V) < |e|$.

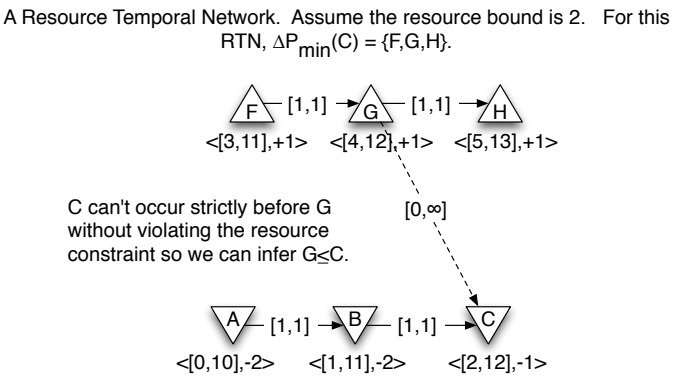

Fig. 4. Inferring new precedence constraints with FBC.

If $|\Delta P^{<}_{max}(X)| = k$ calculating $\sum_{V \in \Delta P^{<}_{max}(X) - S} c(V)$ for every successor or predecessor set requires $O(k^2)$ since we already store the precedences to create the flow graphs. Similar implied precedences can be discovered using the other bounds. To see how this works, consider Figure 4. In this case $\Delta P^{<}_{max}(C) = \{F, G, H\}$. The successor set of G is $\{G, H\}$ and $r + \sum_{V \in \{B(C)\}} = -2$. We see that allowing C to occur before G would lead to a resource constraint violation, since the only event remaining

in $\Delta P^{<}_{max}(C)$ is F; it produces 1, which is not enough to offset $r + \sum_{V \in \{B(C)\}} = -2$. In this case we can infer a new precedence constraint $G \leq C$.

Note that if we relax the assumption that event resource impacts are known and allow schedulers to choose resource impacts, we can also infer restrictions on resource impacts as BC does [6].

4.2 Relating E_t and FBC

In this section, we formally establish the relationship between the intervals over which the FBC and E_t bounds hold. Suppose we find a set $P^{<}_{max}(X)$ that supports $L^{<}_{max}(X)$. What is the interval of time over which this value holds? $L^{<}_{max}(X)$ is the maximum available resource before X occurs assuming that we impose the constraints $\forall V \in P^{<}_{max}(X)\{V < X\}$ and $\forall V \in P^{<C}_{max}(X)\{X \leq V\}$ to get a new RTN $\mathcal{R}'$. Note that the new unordered set $U'(X) = \emptyset$, and $BS'(X) = \emptyset$, and the new closed set $B'(X) = \{P^{<}_{max}(X) \cup B(X)\}$. $L^{<}_{max}(X)$ can be the resource availability no earlier than $V^{*}_{lb} = \max_{V \in B'(X)} V_{lb}$. Forcing events to precede X does not change their earliest start times. However, forcing events to follow X may lead to a new upper bound for X, $X_{ub'}$. Thus, $L^{<}_{max}(X) = L_{max'}(t)$ over the interval $[V^{*}_{lb}, X_{ub'} - 1]$. An example is shown in Figure 5.

Now suppose we find a set $P^{>}_{max}(X)$ that supports $L^{>}_{max}(X)$. This bound assumes that we impose the constraints $\forall V \in P^{>}_{max}(X)\{V \leq X\}$ and $\forall V \in P^{>C}_{max}(X)\{X < V\}$ to get a new RTN $\mathcal{R}'$. Note that the new pending set $U'(X) = \emptyset$, and $\{B'(X) \cup S'(X) \cup BS'(X)\} = P^{>}_{max}(X)$. $L^{>}_{max}(X)$ can be the resource availability no later than $W^{*}_{ub} = \min_{W \in P^{>C}_{max}(X)} W_{ub}$. Forcing events to follow X does not change their upper bounds. However, forcing events to precede X may lead to a new lower bound $X_{lb'}$. that $L^{>}_{max}(X)$ holds. Thus, $L^{>}_{max}(X) = L_{max'}(t)$ over the interval $[X_{lb'}, W^{*}_{ub} - 1]$.

We now can demonstrate the relationship between $L_{max}(t)$ and the bounds $L^{<}_{max}(X), L^{>}_{max}(X)$ and $L_{min}(t)$ and the bounds $L^{<}_{min}(X), L^{>}_{min}(X)$. The simplest result would be to prove that one of the event bounds holds at all times and show how to construct $L_{max}(t)$ from the event bounds; unfortunately, we do not prove this. However, we can generate the entire time-based profile $L_{max'}(t)$ after imposing the temporal constraints for each event bound and use this to show a resource bound for all times.

Theorem 3. *If $L^{<}_{max}(X)$ holds over the interval $[x, y]$ then $\forall t \in [x, y] L^{<}_{max}(X) \leq L_{max}(t)$. If $L^{>}_{max}(X)$ holds over the interval $[x, y]$ then $\forall t \in [x, y] L^{>}_{max}(X) \leq L_{max}(t)$. If $L^{<}_{min}(X)$ holds over the interval $[x, y]$ then $\forall t \in [x, y] L^{<}_{min}(X) \leq L_{min}(t)$. If $L^{>}_{min}(X)$ holds over the interval $[x, y]$ then $\forall t \in [x, y] L^{>}_{min}(X) \leq L_{min}(t)$.*

Proof. The paragraph above describes the intervals over which $L^{<}_{max}(X)$ and $L^{>}_{max}(X)$ hold. By introducing temporal constraints we only eliminate feasible schedules; the rest follows immediately from the definition of $L_{max}(t)$ and Theorems 1 and 2. The proofs for the other event-centric bounds are similar. □

5 Dominance

In this section we will describe our dominance criteria and show that FBC dominates E_t and BC; we do not formally show that BC and E_t do not dominate each other, relying on the intuition of Section 3 to adequately demonstrate this. In order to motivate our definition of dominance, suppose that some procedure A is used to check the HC in a chronological search framework with a static variable and value ordering heuristic. Then we would like A to dominate B if A leads to smaller search trees than using B.

Definition 1. *Let R be an RTN. Let $T(R)$ be the set of all temporally consistent RTNs that can be formed from R by adding temporal constraints to R. Let $A, B : T(R) \rightarrow \{T, F, ?\}$ Let $U_A(R) = \{S \in T(R) | A(S) = ?\}$. Then A dominates B on R if $U_A(R) \subset U_B(R)$. A dominates B if $\exists R$ such that A dominates B on R and there exists no S such that B dominates A on S. We write $A \prec_R B$ or $A \prec B$ as appropriate.*

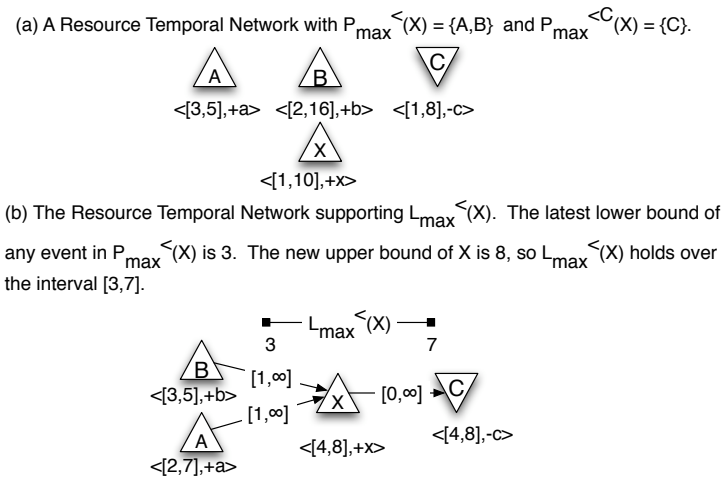

Fig. 5. Deriving the time intervals over which resource availability bounds hold. The example shows the calculation for $L^{<}_{max}(X)$.

Theorem 4. $FBC \prec BC$.

Proof. Theorems 1 and 2 show that the flow construction guarantees the tightest possible bounds on $L^{<}_{max}(X), L^{>}_{max}(X), L^{<}_{min}(X)$ and $L^{>}_{min}(X)$ for *any* RTN. Thus there can be no S such that $BC \prec_S FBC$. The example shown in Figure 1 shows at least one RTN R for which $FBC \prec_R BC$. This completes the proof. □

Theorem 5. $FBC \prec E_t$

Proof. Theorems 1, 2 and 3 shows that there is no RTN S for which $E_t \prec_S FBC$. The example in Figure 2 shows that there is at least one RTN R for which $FBC \prec_R E_t$. This completes the proof. □

6 Complexity of Calculating FBC

A naive algorithm for calculating FBC builds a new flow network for each event $X \in \mathcal{N}$. An equally naive analysis of the complexity of this algorithm is $O(n * MaxFlow(n,m) + n^2)$, where m is the number of anti-precedence relationships in the RTN. However, this analysis is unsatisfactory for a number of reasons. Consider the RTN in Figure 1. The flow networks required to calculate FBC are *identical* for A, B, C, D because $S(A) = S(B) = S(C) = S(D)$. Additionally, the flow networks include only a small fraction of the n nodes in the RTN; strict precedences and equalities among events will generally reduce the size of the flow problems. These observations suggest it might be possible to improve upon the bound of $O(n*MaxFlow(n,m)+n^2)$ to calculate FBC.

6.1 A Nontrivial Lower Complexity Bound

In this section we provide a *lower* bound on the complexity of the naive approach to calculating FBC. We do so by constructing an RTN such that the flow problem to solve for each event is both non-trivial and distinct. The RTN is a "square" graph with $\sqrt{n}$ events per side. We index events by row and column in the square. The RTN has the following strict precedences: $(i,j) < (i+1,j)$, and $(i,j) < (i,j+1)$ (obviously omitting those links for which the indices are outside the bounds $[0,\sqrt{n}]$. By construction, $P((i,j)) = ((x,y)|x > i \wedge y < j) \cup ((u,v)|u < i \wedge v > j)$. Thus, all of the flow graphs

are distinct and nontrivial. Notice that we can assign $c(X)$ arbitrarily to the events of the RTN, as long as they are all non-zero and there is a mix of consumers and producers. This RTN is shown in Figure 6.

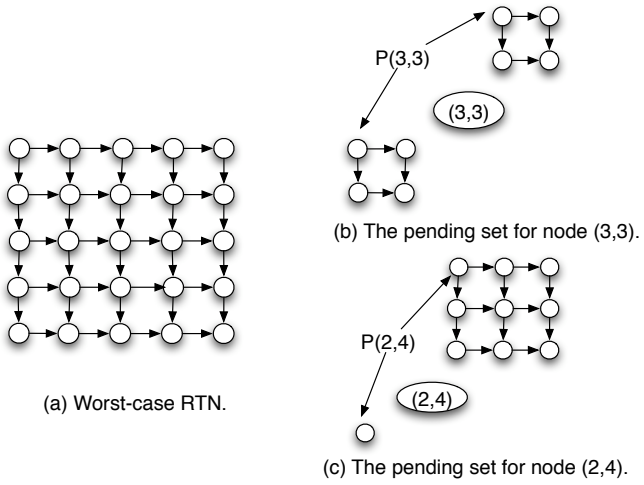

Fig. 6. A perverse RTN providing a worst-case lower bound for calculating FBC.

We now proceed to construct a lower bound on the complexity of the naive approach for calculating FBC on this RTN. By construction we have guaranteed that no "quick fixes" can be used to decrease the complexity. The larger of the two induced flow problem for event (i, j) contains $\max((i-1)(\sqrt{n}-j-1), (j-1)(\sqrt{n}-i-1))$ events, and at least this many flow arcs (we could do an exact count but it isn't necessary since we're providing a lower bound.) Let us now assume we are using a FIFO preflow-push algorithm to solve each flow problem [10]; this ignores any efficiency gained from analyzing the pushable flow, but is also suitable for our purposes. If v is the number of nodes in the flow problem, there are at least v edges, and so the complexity of solving the flow problem is $\Omega(v^{2.5})$. Using these assumptions the total complexity of solving all the flow problems is

$$\leq \sum_{j=1}^{\sqrt{n}} \sum_{i=1}^{\sqrt{n}} \max((i-1)(\sqrt{n}-j-1), (j-1)(\sqrt{n}-i-1))^{2.5}$$

First, we simplify the sum to get a lower bound:

$$\leq \sum_{j=0}^{\sqrt{n}-1} \sum_{i=0}^{\sqrt{n}-1} (i(\sqrt{n}-j))^{2.5}$$

We next approximate the sum with the integral (which, while bounding above, is close enough for our purposes):

$$\approx \int_{j=0}^{\sqrt{n}-1} \left(\int_{i=0}^{\sqrt{n}-1} (i(\sqrt{n}-j))^{2.5} di \right) dj$$

The first integral with respect to i is trivial. For the second integral we have

$$\int (\sqrt{n}-j)^{2.5} dj = (\sqrt{n}-j)^{2.5} \left(\frac{j-\sqrt{n}}{3.5} \right)$$

Collecting terms we have

$$\sum_{j=1}^{\sqrt{n}}\sum_{i=1}^{\sqrt{n}} \max((i-1)(\sqrt{n}-j-1),(j-1)(\sqrt{n}-i-1))^{2.5} \leq \left(\frac{(\sqrt{n}-1)^{3.5}}{3.5}\right)\left(\frac{\sqrt{n}^{3.5}}{3.5}-\frac{1}{3.5}\right)$$

Collecting the high order positive powers of $\sqrt{n}$ we see that the naive algorithm has a lower bound $\Omega(n^{3.5})$. Thus, for preflow-push flow algorithms using FIFO queues, the naive algorithm for FBC requires $\Omega(n * MaxFlow(n,m))$ for solving the flows.

6.2 Incrementally Calculating FBC

As stated previously, the incremental technique described in [7] shaves a factor of n off the cost of calculating E_t. To do so, the incremental approach relies on restricted modifications of the flow problem. If A is in the flow problem, it may be removed if it has no successors other than the sink remaining in the flow problem; if B is not in the flow problem, it may be added if it has no predecessors other than the source in the flow problem. This provides some hope that we can find a way to eliminate the factor of n "extra" cost for calculating FBC that we described in the previous section. Unfortunately, this is not the case. The crucial element of the complexity analysis in [7] requires that an event always move from open to pending to closed. We show that naively applying the incremental algorithm may result in a non-trivial number of events moving from closed to pending, thereby defeating the cost-savings measures. Consider the RTN described in Figure 6. The longest chain of events is $O(\sqrt{n})$, and there are $O(\sqrt{n})$ chains of events we must solve for which the incremental approach cannot save any computation. Each of these induces a total complexity of $O(MaxFlow(n,m))$. Thus, for this problem, even the incremental approach to solving flow problems customized for the E_t algorithm cannot reduce the complexity of FBC below $\Omega(\sqrt{n} * MaxFlow(n,m) + n^2)$.

We can take advantage of the incremental flow calculation by judicious ordering of the bounds we calculate for an event X. For example, to find $\Delta P^{<}_{max}(X)$ we solve a flow problem over $\{BS(X) \cup U(X)\}$. To find $\Delta P^{>}_{max}(X)$ we solve a flow problem over $\{AS(X) \cup U(X)\}$. The change in the flow problem to be solved allows the incremental approach to be used; thus, we can use the solution of the flow that gives us $\Delta P^{<}_{max}(X)$ in order to find $\Delta P^{>}_{max}(X)$. The total complexity of solving both flow problems is $O(MaxFlow(n,m))$ where n and m are derived from the anti-precedence graph on $\{BS(X) \cup U(X) \cup AS(X)\}$; this is cheaper than solving both flows separately as long as $U(X) \neq \emptyset$.

We can also take advantage of the incremental flow calculation by storing and reusing flow calculations and ordering the bounds calculations for different events. Suppose $Z \in \{BS(X) \cup S(X)\}$, we are calculating the bounds for X and Z's bounds were previously calculated. Then $A \in BS(Z)$ must be either in $B(X)$ or $BS(X)$ and $B \in U(Z)$ must be either in $B(X), BS(X)$ or $U(X)$. This fits the conditions required to use the incremental approach by reusing the solution to the flow problem on $\{U(Z) \cup BS(Z)\}$. If we had *cached* this solution, we could then retrieve it and reuse it in time proportional to the number of previously stored flow calculations. We can generate an event order $\Pi(X)$ using a Depth-First Search through the anti-precedence graph on all of the events to ensure that we can take advantage of incremental calculations. We call the resulting algorithm $FBC - DFS$, and it is described in Figure 7 below. As before, we need to calculate bounds for only one event $X \in S(Y)$.

Obviously, the cost of $FBC - DFS$ depends on the DFS traversal used to order the bounds calculations, as described in the next result.

Theorem 6. *Let $\mathcal{R}$ be an RTN with m induced anti-precedence constraints in its STN $\mathcal{S}$. Let Π be an ordering induced by a DFS traversal of the anti-precedence graph on $\mathcal{S}$, let w be the number of leaves of the traversal and let l be longest path in the traversal. Then Algorithm $FBC - DFS$ takes $O(w * MaxFlow(n,m)) + n^2$ time, and takes $O(lm)$ space.*

FBC-DFS($\mathcal{R}, \Pi(X)$**)**
Enforce Arc-Consistency on $\mathcal{R}$
Maintain event-centric partitions
$F_{max} = F_{min} = \emptyset$
for each event $X_i \in \Pi(X)$
Find the latest $X_j \in \Pi(X)$ such that $j < i, X_j \in B(X_i) \cup BS(X_i) \cup S(X_i)$
if $X_j \neq X_{i-1}$ Pop F_{max} and F_{min} to X_j
Inc-Bound(X_i)
end for
end

Inc-Bound(X**)**
Build flow problems for $L^{<}_{max}(X)$ from $top(F_{max})$ and $L^{<}_{min}(X)$ from $top(F_{min})$
Find $\Delta P^{<}_{max}(X) \subset \{BS(X) \cup U(X)\}$
$L^{<}_{max}(X) = \sum_{V \in \{\Delta P^{<}_{max}(X) \cup B(X)\}} c(V)$
Find $\Delta P^{<}_{min}(X) \subset \{BS(X) \cup U(X)\}$
$L^{<}_{min}(X) = \sum_{V \in \{\Delta P^{<}_{min}(X) \cup B(X)\}} c(V)$
Build flow problems for $L^{>}_{max}(X), L^{>}_{min}(X)$
Find $\Delta P^{>}_{max}(X) \subset \{AS(X) \cup U(X\})$
$L^{>}_{max}(X) = \sum_{V \in \{\Delta P^{>}_{max}(X) \cup B(X) \cup BS(X) \cup S(X)\}} c(V)$
Find $\Delta P^{>}_{min}(X) \subset \{AS(X) \cup U(X)\}$
$L^{>}_{min}(X) = \sum_{V \in \{\Delta P^{>}_{min}(X) \cup B(X) \cup BS(X) \cup S(X)\}} c(V)$
Push the flow solutions for $L^{<}_{max}(X)$ onto F_{max} and $L^{<}_{min}(X)$ onto F_{min}
end

Fig. 7. A sketch of the Flow Balance Constraint implemented by using a Depth-First search through the precedence graph and using an incremental approach to reduce computation time for constructing the bounds.

Proof. Each leaf corresponds to a sequence such that $X_{i-1} \in \{B(X_i) \cup BS(X_i) \cup S(X_i)\}$. Then, for each sequence, we can reuse the flow solutions from $top(F_{max})$ and $top(F_{min})$ to solve each X_i; each chain costs $O(MaxFlow(n, m))$. Since there are w such chains, we're done with this part of the proof. Since the longest chain is of length l, we store at most l copies of a flow problem on at most n nodes and m edges.

The theory of partially ordered sets gives some bounds on w and l; Let W be the partial order width and L be the partial order length of the partially ordered set induced by the anti-precedence graph. For any DFS traversal, $w \geq W$ and $l \leq L$. However, it is not obvious in general how to efficiently find a good event ordering.

7 Higher Order Balance Constraint

The quantities $L^{<}_{max}(X), L^{>}_{max}(X), L^{<}_{min}(X)$ and $L^{>}_{min}(X)$ can be thought of as *first order* bounds on resource availability, in that they represent resource bounds before and after one event. In this section we generalize these techniques in order to calculate higher-order resource availability checks after k events. To see why this is valuable, consider Figure 8. Initially there are 2 units of resource available. We see that no first order bound calculated by FBC proves that this RTN is impossible to solve. However, notice that we can show that the *maximum* available resource after both A and B have occurred, which we denote $L^{>}_{max}(A \wedge B)$, is -1. This corresponds to one of two schedules: it is possible to either schedule $D \leq B$ or $C \leq A$, but not both simultaneously. Thus, without further search, we can prove that the HC is satisfied because no solution exists.

This example shows that it may be valuable to perform 2^d-order checks on resource availability by determining resource availability immediately before and after

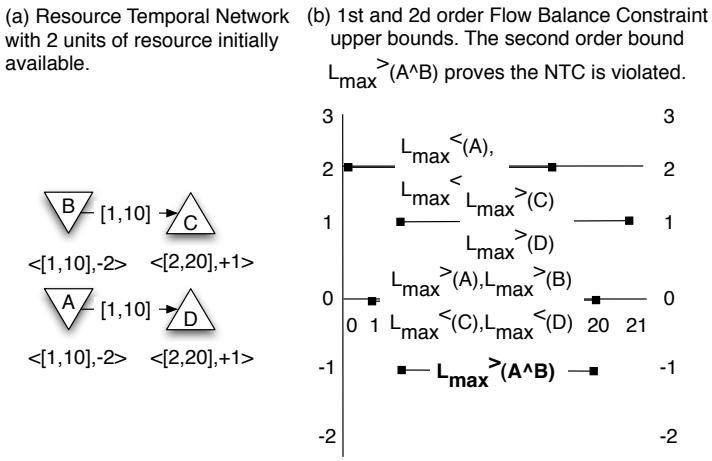

Fig. 8. An RTN for which FBC fails to prove that the HC is unsatisfied.

sets of 2 events. We assume without loss of generality that $X \in U(Y)$. In order to do this for $L^{<}_{max}(X \wedge Y)$ we must account for the following possibilities: neither X nor Y have occurred, X has occurred but Y has not, and vice versa. To find the maximum availability strictly before both X and Y we solve the flow problem over the events of $\{\{BS(X) \cup U(X)\} \cap \{BS(Y) \cup U(Y)\}\}$. We call the resulting set of events $\Delta P^{<}_{max}(X * Y)$. To find the maximum availability strictly before Y assuming $X < Y$ requires adding the constraint and recalculating the partitions of the anti-precedence graph relative to Y. We then solve the flow problem defined by $BS'(Y) \cup U'(Y)$ and call the resulting set $\Delta P^{<}_{max}(X < Y)$. To find the maximum availability strictly before X assuming $Y < X$ proceeds similarly, and the resulting set is called $\Delta P^{<}_{max}(Y < X)$. We define $L^{<}_{max}(X \wedge Y)$ as

$$\max\Big(\sum_{V \in \{\Delta P^{<}_{max}(X*Y) \cup B(X) \cap B(Y)\}} c(V) \sum_{V \in \{B'(Y) \cup \Delta P^{<}_{max}(X<Y)\}} c(V) \sum_{V \in \{B'(X) \cup \Delta P^{<}_{max}(Y<X)\}} c(V)$$

and $\Delta P^{<}_{max}(X \wedge Y)$ is the set of events defining the maximum.

For $L^{>}_{max}(X \wedge Y)$ we we must account for the following possibilities: X and Y occurred at the same time, X occurred before Y, and vice versa. These options account for the possibility that, for example, events in $A(X) \cap U(Y)$ contribute to $L^{>}_{max}(X \wedge Y)$. To find the maximum assuming $X = Y$, we first solve the flow problem over the events of $\{\{AS(X) \cup U(X)\} \cap \{AS(Y) \cup U(Y)\}\}$. We call the resulting set $\Delta P^{>}_{max}(X = Y)$. To find the maximum assuming $X < Y$ requires adding the constraint and recalculating the partitions of the anti-precedence graph relative to Y. We solve the flow problem over $\{AS'(Y) \cup U'(Y)\}$, and call the resulting set $\Delta P^{>}_{max}(X < Y)$. and $\Delta P^{>}_{max}(X > Y)$ is defined similarly. We define $L^{>}_{max}(X \wedge Y)$ as

$$\max\Big(\sum_{V \in (\Delta P^{>}_{max}(X=Y) \cup B(X) \cup BS(X) \cup S(X) \cup B(Y) \cup BS(Y) \cup S(Y)} c(V)$$

$$\sum_{V \in B'(Y) \cup BS'(Y) \cup S'(Y) \cup \Delta P^{>}_{max}(X<Y)} c(V)$$

$$\sum_{V \in B'(X) \cup BS'(X) \cup S'(X) \cup \Delta P^{>}_{max}(Y<X)} c(V)$$

and $\Delta P^{>}_{max}(X \wedge Y)$ is the set of events defining the maximum. The lower bounds are calculated in a similar manner.

Let us see how this works in Figure 8. To find $L^{<}_{max}(A \wedge B)$, note $\{\{BS(A) \cup U(A)\} \cap \{BS(B) \cup U(B)\}\} = \emptyset$ and $\{B(A) \cap B(B)\} = \emptyset$. Thus, $L^{<}_{max}(A * B) = 2$, which is achieved by assuming no event has taken place. To find $L^{<}_{max}(A < B)$, we see that $U'(B) = D$; the best schedule here is A, D for an availability of 1. Similarly, to find $L^{<}_{max}(B < A)$, we see that $U'(A) = C$; the best schedule here is B, C for an availability of 1. Thus, $L^{<}_{max}(X \wedge Y) = 2$. To calculate $L^{>}_{max}(A \wedge B)$ we see that $\Delta P^{>}_{max}(A = B) = \emptyset$, leading to $L^{>}_{max}(A = B) = -2$. However, $\Delta P^{>}_{max}(A < B) = D$; this corresponds to the schedule A, D, B, with the result that $L^{>}_{max}(A < B) = -1$, and symmetrically $L^{>}_{max}(B < A) = -1$. Thus, $L^{>}_{max}(A \wedge B) = -1$.

The total complexity of the resulting naive algorithm for calculating *Second order Flow Balance Constraint* (**FBC^2**) is $\Omega((n^2) * (MaxFlow(n, m) + n^2))$. The n^2 term comes from the fact that $n(n-1)$ pairs of bounds must be calculated. The complexity bounds obscure the fact that 6 flow problems must be solved per pair of events, and also hides the fact that the precedence constraints must be recalculated for each constraint imposed to set up the flow problems. Note, however, that n^2 is a very crude estimate of the total number of bounds to compute. If A strictly precedes B then $P^{<}_{max}(A \wedge B) = P^{<}_{max}(B)$. Thus, the induced precedences vastly reduce the number of bounds to calculate. Additionally, the sizes of the flow problems will generally be larger as the number of events involved climbs. These factors make a more precise complexity analysis difficult. Finally, it is likely that the incremental flow algorithm described in [7] can be used to further reduce the complexity. It is sufficient for our purposes to demonstrate that even tighter inferred constraints can be calculated in time polynomial in the number of events considered. While we can generalize further to higher order bounds, the apparent complexity even of the second order bounds makes it likely that these higher order bounds will not be practical to compute.

8 Conclusions and Future Work

In this paper we have also shown how to exploit the features of BC and E_t to construct FBC, a tighter bound on the availability of resources for RTNs than either of the previous approaches. We have shown that FBC dominates both E_t and BC in terms of the size of search trees generated under chronological backtracking search with a static variable ordering, and that contrary to expectations, BC and E_t do not strictly dominate each other. We have described an incremental algorithm, $FBC - DFS$ that takes advantage of incremental flow calculations by using a depth-first search traversal of the anti-precedence graph. If w is the number of leaves of the traversal and l is longest path in the traversal, algorithm $FBC - DFS$ takes $O(w * MaxFlow(n, m))$ time, and takes $O(lm)$ space. The technique generalize for calculating FBC leads to even tighter bounds, but at sharply increased computational cost.

While we have proven dominance of FBC, an empirical study will be necessary to determine the relative value of E_t, BC and FBC for speeding up the solving of scheduling problems. In particular, it is necessary to determine how to generate good orderings to optimize the performance of $FBC - DFS$. A second, equally important empirical study will be necessary to shed light on how to integrate heuristics that make use of the various bounds. Laborie [6] has built numerous such heuristics for BC. However, such heuristics have complex interactions with the pruning power of the envelopes. It will likely be necessary to trade off between the pruning power and heuristic predictiveness of the resource bounds to craft the best scheduling algorithm.

Changes to dominance criteria that we use are worth contemplating. A dominance criteria for dynamic variable ordering based search would be handy but is complicated due to the interaction of pruning techniques and heuristics. Dominance criteria for local search based algorithms are also desirable. On the one hand, requiring that A dominates B on R if $U_A(R) \subset U_B(R)$ is rather strong, and could be weakened, say, to $|U_A(R))| < |U_B(R))|$. On the other hand, requiring that A dominates B if $\exists R$ such that A dominates B on R and there exists no S such that B dominates A on S is somewhat weak, and perhaps could be strengthened.

9 Acknowledgments

I would like to thank Ari Jónsson and Nicola Muscettola for numerous discussions on this topic. This work was funded by the NASA Intelligent Systems Program.

References

1. Muscettola, N.: Computing the envelope for stepwise-constant resource allocations. In: Proceedings of the 8^{th} International Conference on the Principles and Practices of Constraint Programming. (2002) 139 –154
2. Laborie, P.: Resource temporal networks: Definition and complexity. In: Proceedings of the 18^{th} International Joint Conference on Artificial Intelligence. (2003) 948 – 953
3. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991) 61–94
4. Jónsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in interplanetary space: Theory and practice. In: Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling. (2000)
5. Morris, P., Muscettola, N., Tsamardinos, I.: Reformulating temporal plans for efficient execution. In: Proceedings of the 15^{th} National Conference on Artificial Intelligence. (1998)
6. Laborie, P.: Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. Artificial Intelligence **143** (2003) 151–188
7. Muscettola, N.: Incremental maximum flows for fast envelope computation. In: Proceedings of the 14^{th} International Conference on Automated Planning and Scheduling. (2004)
8. Chapman, D.: Planning for conjunctive goals. Artificial Intelligence **32** (1987) 333 – 377
9. Kumar, T.K.S.: Incremental computation of resource-envelopes in producer-consumer models. In: Proceedings of the 9^{th} International Conference on the Principles and Practices of Constraint Programming. (2003) 664 – 678
10. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice Hall (1993)